Phœnix: Reviving MLC Blocks as SLC to Extend NAND Flash Devices Lifetime

Xavier Jimenez, David Novo and Paolo Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH–1015 Lausanne, Switzerland
Email: {xavier.jimenez, david.novobruna, paolo.ienne}@epfl.ch

Abstract—**On a *Multi-Level Cell* (MLC) flash memory, a flash block that is becoming unreliable to store multiple bits per cell can be "revived" by storing only a single bit per cell. While the revived-block capacity is halved, its lifetime is significantly extended without jeopardizing the stored data. We present Phœnix, a technique that benefits from this feature to extend a device lifetime, and we evaluate its potential through detailed trace simulation on realistic benchmarks. Phœnix shows systematic lifetime extensions ranging from 3% up to 17%, without extra memory requirements or performance loss.**

I. Introduction

NAND flash technology features low cost, low energy, high mobility and high performance, making it the storage media of choice for handheld devices. However this technology comes with its share of drawbacks; flash cells require to be erased before being written (*programmed*). Furthermore, the erase process is applied at a coarser granularity (*blocks*) than the read and program accesses (*pages*), generally enforcing impractical out-of-place updates. Lastly, and more importantly, flash cells have a limited endurance: they experience high stress during the program and erase processes, which degrades them gradually and renders them eventually unreliable.

As continuous pressure is put on the technology to develop memories with higher densities and, hence, a smaller cost per bit, flash cells grow multi-bit. Nowadays, *Multi-Level Cell* (MLC) technology, where a single flash cell is able to store 2 bits, is well established. Unfortunately, this increase in bit density comes with lower performances and with a severe hit in reliability, as programming and reading become more complex [1]. In all practical applications, the severity of those limitations is somehow mitigated by a software abstraction layer, called *Flash Translation Layer (FTL)*. An FTL translates logical to physical addresses and provides a simpler interface, hiding physical aspects and constraints of flash. It typically deals with garbage collection and wear-leveling.

In this paper, we present Phœnix, a novel scheme to extend current FTL that mitigate the degradation in lifetime of MLC flash. We propose to keep on using worn-out MLC blocks as SLC blocks. By "reviving" these blocks, which would be normally considered as useless, we show that the lifetime of current flash devices can be extended by up to 17%.

978-3-9815370-0-0/DATE13/©2013 EDAA

II. Reviving Bad Blocks

Flash memory technology degrades with use, and thus, its endurance is typically expressed in *Program/Erase* (P/E) cycles. Accordingly, flash memory devices are warranted to be functional up to a certain number of such cycles. Programming and reading a single bit in a cell is more reliable than multiple bits. Thus, *Single-Level Cell* (SLC, 1 bit per cell) can generally experience one and two order of magnitude more P/E cycles than MLC (2 bits per cell) and TLC (3 bits per cell), respectively [1].

Interestingly, a typical MLC can also be managed as SLC by storing a single bit [1], [2]. Thereby, the MLC experiences the performance, energy and endurance benefits of SLC technology at expenses of capacity. Throughout the rest of the paper, this will be referred as the *SLC-mode* of an MLC.

When a cell becomes unreliable for MLC use, it can still be used in SLC-mode for a significant amount of extra P/E cycles. In this paper, we propose a method using this feature to extend the lifetime of flash storage devices that relies on a hybrid FTL.

A. Baseline Architecture: Hybrid FTLs

Hybrid-FTL architectures [3]–[7] combine page-level and block-level mappings. The fine-grained page-level mapping brings more flexibility but comes with a large mapping table compared to the coarser block-level mapping. Thus, a hybrid mapping balances both factors by dividing the flash memory into two regions: a large data partition addressed at the block level, and a small log-buffer partition addressed at the page level. The purpose is to direct random writes to the log buffer so that they can be written back to the data partition in order as big chunks. Thereby, the garbage collection overhead can be kept low for a reasonable mapping-table memory (e.g., SRAM, DRAM) requirement.

Considering that a significant amount of writes are directed to the small buffer partition, previous work [8]–[10] proposed to build the buffer partition on SLC flash, which provides high performance but poor density, and the larger data partition on MLC of lower performance but higher density. As a result, the flash device exhibits performances close to SLC (particularly for random data accesses) while keeping the area efficiency of MLC to a great extent.

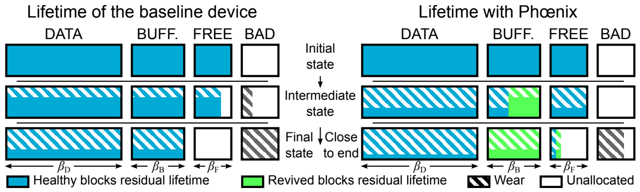

Fig. 1. **Phœnix compared to the baseline architecture.** Each architecture starts with a data, a buffer and a free set of size β_D, β_B and β_F, respectively and a bad set initially empty. While the baseline approach discards unreliable blocks to the bad set, Phœnix revives them as SLC and uses them in the buffer. Reviving a bad block extends its lifetime and reduces the stress on the remaining healthy blocks. In the end, Phœnix will benefit SLC-mode lifetime, while better exploiting the MLC lifetime.

Interestingly, similar benefits can also be achieved in storage systems built only on MLC chips where the buffer partition is managed in SLC-mode [2]. We consider this system as our baseline architecture.

B. Reviving MLC Blocks in SLC-mode

Typical flash devices stop using a block whenever it permanently fails or when it becomes unreliable by showing a bit error rate too high to be handled by the implemented *Error Correcting Code* (ECC). For this reason, flash manufacturers suggest to reserve at least 2% of the total capacity [11] for free (or spare) blocks to replace the “bad” ones. Such a device will die when running out of free blocks.

While a permanent failure prevents any further use of a block, we propose to revive the unreliable blocks with high error rates by using them in SLC-mode. Revived blocks will be allocated to the buffer partition, where blocks typically take a greater amount of incoming writes than data blocks. Thereby, revived blocks are not only sparing the use of free blocks, they also reduce the stress on remaining healthy blocks.

Revived blocks can be managed in a very similar way to typical bad block management [11]. A list of bad block is generally maintained in some reserved area of the flash device and can easily be extended with a flag to differentiate bad blocks from revived one, at practically no cost. A similar flag could redundantly reside in the FTL memory for every block allocated to the buffer or free set to quickly differentiate an healthy block from a revived one, and thus, avoiding any identification-time overhead.

Fig. 1 illustrates the difference between a typical baseline architecture and the proposed Phœnix technique during their lifetime. Both initially allocates four different partitions: a data partition of size β_D, a buffer partition of size β_B managed in SLC-mode, a set of β_F free blocks, and a set of bad blocks that we assume empty at the beginning of the device life. Throughout its lifetime, the baseline architecture will regularly identify blocks as broken or unreliable and move them to the bad set, gradually emptying the free set. When the device runs out of free blocks, it is considered dead. This is illustrated in Fig. 1 by the last state of the baseline example (bottom left).

When using Phœnix, the buffer and the free set can now allocate both revived and healthy blocks, while the data partition is still restricted to healthy blocks. Blocks that are detected as unreliable for MLC are labeled as *revived* and are kept in the free set. (Permanently broken blocks are directly moved to the bad set.) Fig. 1 shows how healthy blocks get progressively replaced by revived blocks in the buffer. Now, less healthy blocks are required for the device to stay alive which results in a longer device lifetime.

With Phœnix, the buffer would ideally allocate only revived blocks maintaining the device alive as long as enough healthy blocks are available for the data partition. Whenever the buffer needs to allocate a new block from the free set, it will give priority to the revived blocks in order to minimize the stress on the healthy blocks. Thus, a block will only be dropped into the bad set when it is considered to be unreliable in SLC-mode. In the next section, we discuss in detail the models used to evaluate quantitatively the lifetime improvements that can be expected from Phœnix.

III. Device Degradation Models

We measure empirically the block endurance distribution from real NAND flash chips. Based on this data, we propose two models to describe and confront the lifetime of a baseline device against Phœnix.

A. Block Endurance Distribution

In order to measure how P/E cycles affect the different blocks in the device, we run a synthetic test on a set of 50 blocks for a couple of MLC NAND flash chips. The test repeats the following cycle: each block is erased and then programmed with random data, which is read back and checked for correctness. The test monitors the evolution of erroneous bits per page for several thousands of cycles.

A particular block is considered to be unreliable whenever a given error threshold is reached in any of its pages. The actual threshold will depend on the ECC capabilities of a particular device. A stronger ECC extends significantly the lifetime of a block but requires more complex hardware, increases access latency and adds some storage overhead. Accordingly, Fig. 2 plots the block endurance cumulative distribution measured in one of our chips for 4- and 8-bit error thresholds. The chip is organized in blocks made of 128 pages of 8 kBytes. The distribution curves for our other NAND flash chips vary mainly in the average endurance, but keeps the same distribution shape.

We fit the measured endurance distribution with the following inverse hyperbolic tangent function:

$$f(\rho) = a \cdot \operatorname{artanh}(2 \cdot \rho - 1) + b \quad \text{for} \quad 0 < \rho < 1, \qquad (1)$$

with $f(\rho)$ being the largest endurance in P/E cycles within the ρ weakest blocks, and a and b being the parameters of the distribution. Parameter b corresponds to the average endurance, while a is function of the variance. We provide the parameter set for each fitted curve in Fig. 2.

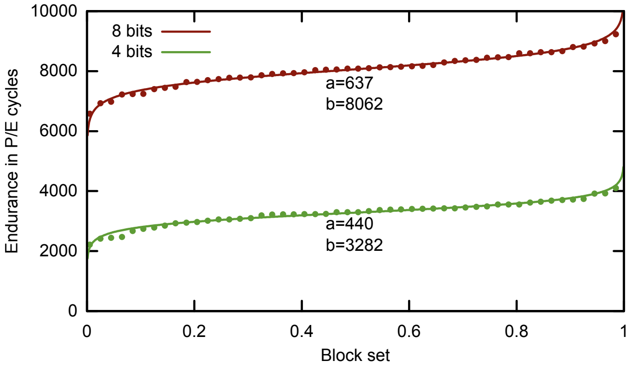

Fig. 2. **Block endurance measured in a real NAND flash chip.** We report the endurance of a set of blocks measured from a real NAND flash chip, assuming four and eight faulty bits as error thresholds. The blocks are ordered from the smallest endurance (on the left) to the largest (extreme right). Each solid line is the model function of Equation 1 fitted to the measured data (markers). For each, we provide the corresponding parameters a and b.

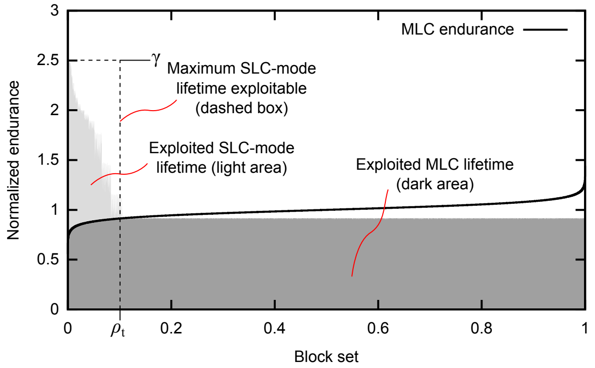

Fig. 3. **Lifetime breakdown.** The solid line shows the block MLC endurance normalized to the average. The dark region represents the exploited MLC lifetime given a maximum ρ_t bad blocks. The light region is an example of the additional SLC-mode lifetime that could be exploited by Phœnix. Using the maximum SLC-mode lifetime delimited by the dashed box is challenging, because, as more blocks get revived, the time to exploit them gets shorter.

B. Analytical Model of Baseline Device Lifetime

Next, we present an analytical model to compute the expected lifetime of a typical flash device given its block endurance distribution and it partitions size. We refer to lifetime as the total amount of data that can be written in a block or a device before wearing it out. In a previous work [2], we studied the MLC- and SLC-mode mixed usage and observed empirically that programming a block once in MLC or twice in SLC-mode applies practically the same wear to the block. Conservatively, we assume an equal wear per written bit for SLC- and MLC-mode.

Let's ρ_t be the maximum ratio of blocks that can wear out for a flash device before it dies. The integral of $f(\rho)$ from 0 to ρ_t gives us the lifetime exploited by weakest blocks, while the remaining healthy blocks are limited to $f(\rho_t)$ cycles. Hence, the MLC lifetime component L_{MLC} of a flash device is

$$L_{\mathrm{MLC}}(\rho_t) = [F(\rho)]_0^{\rho_t} + f(\rho_t)\cdot(1-\rho_t). \qquad (2)$$

This lifetime is illustrated in Fig. 3 by the surface of the dark area. The model assumes a perfect wear-leveling algorithm that evens the P/E counts of every healthy blocks.

The baseline device reaches its lifetime limit when β_{F} blocks wear out. As its lifetime is limited to the MLC lifetime, it is equal to L_{MLC} with $\rho_t = \beta_{\mathrm{F}}/\beta$, and with β being the total amount of block in the device.

C. Analytical Upper Bound of Phœnix Device Lifetime

In order to evaluate the lifetime extension brought by Phœnix, we must describe a relationship between the lifetime of MLC and SLC-mode. We use a parameter γ such that $\gamma \cdot l_{\mathrm{M}} = l_{\mathrm{S}}$, with l_{M} and l_{S} being the lifetime of MLC and SLC-mode, respectively. While l_{M} is provided in manufacturer datasheets, l_{S} is typically not. However, Im and Shin [9] relate the endurance for a specific flash device that explicitly provides SLC and MLC modes. Specifically, the device blocks could sustain either 10K MLC-erase cycles or 50K SLC-erase cycles. Therefore, a block in SLC-mode could be written 2.5× more than in MLC. Thereby, for our experiments, we assume γ to be 2.5.

While the baseline lifetime is bounded by $\rho_t = \beta_{\mathrm{F}}/\beta$, Phœnix extends this limit with $\rho_t = (\beta_{\mathrm{F}} + \beta_{\mathrm{B}})/\beta$. Furthermore, Phœnix revives the weakest ρ_t blocks, which can now expect a maximum endurance of γ. Hence, the maximum Phœnix lifetime corresponds to the union of the dark region with the dashed box from Fig. 3 and is equal to

$$L_{\mathrm{Pmax}}(\rho_t) = \gamma\cdot\rho_t + f(\rho_t)\cdot(1-\rho_t). \qquad (3)$$

As revived blocks are exclusively allocated to the buffer, when it is underutilized, the extra lifetime provided by the SLC-mode cannot be exploited. We define another bound to the maximal reviving lifetime, $L_{\mathrm{Pbuf}}(\alpha,\rho_t)$, that is function of a ratio α of writes to the buffer with

$$L_{\mathrm{Pbuf}}(\alpha,\rho_t) = \frac{L_{\mathrm{MLC}}(\rho_t)}{1-\alpha}. \qquad (4)$$

This function returns the MLC lifetime plus the total amount of writes to the buffer, which corresponds to the maximum SLC-mode lifetime exploitable. Combined with L_{Pmax}, we get $L_{\mathrm{Pbound}}(\alpha,\rho_t)$, the global upper bound, which is the minimum of both functions,

$$L_{\mathrm{Pbound}}(\alpha,\rho_t) = \min(L_{\mathrm{Pmax}}(\rho_t), L_{\mathrm{Pbuf}}(\alpha,\rho_t)). \qquad (5)$$

This upper bound is plotted in Fig. 4 for two different device configurations that we will discuss later. In practice, this bound is very unlikely to be reached: it disregards any sequentiality constraint in the accesses into the buffer and data partitions. Thus, in the next section we produce the expected lifetime gain from simulations.

IV. Results

In this section, we show how Phœnix behaves on a simulated FTL executing realistic application traces. In order to evaluate Phœnix, we implement ROSE [7], a state-of-the-art hybrid FTL, on a flash simulator that we developed. For the simulator

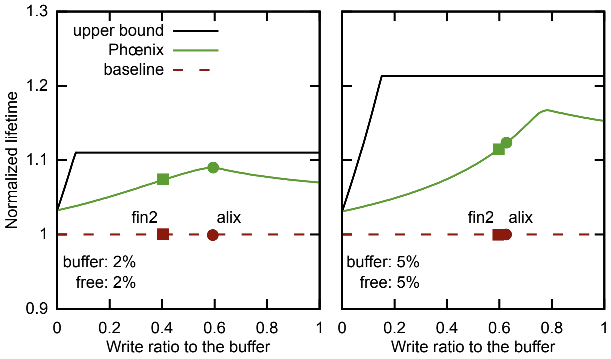

Fig. 4. **Lifetime extension using Phœnix for two configurations.** The configuration for the left figure uses a buffer and a free set of 2% the total capacity each, while the right one 5% each. The lifetime is normalized to the baseline. The upper bound represents the maximal lifetime that could possibly be exploited by Phœnix. Although the efficiency of Phœnix depends on the amount of writes directed to the buffer, it is systematically larger.

input, we use two traces, namely *Financial 2* [12] and *Alix 8d* [2], which show a good balance of large sequential writes with small updates. At the beginning of the simulation, the endurance of every block is randomly set according to the distribution of Equation 1, with parameters $a = 637$ and $b = 8062$. We repetitively input the same trace until the simulated flash device eventually dies. We generate results for two different buffer/free set size (2%/2% and 5%/5%) with and without Phœnix.

Phœnix does not hamper the performance of the FTL, actually it reduces slightly the wear-leveling overhead when approaching the end of the device lifetime, which increases marginally the performance by less than 1%. From our simulations output, we report the ratio of writes to the buffer and the average amount of time that each block was programmed. The generated points are plotted in Fig. 4, while the curves were generated from simplified simulations assuming a constant ratio of write to the buffer. We systematically measured an error lower than 0.1% between the full and simple simulation. Therefore, for a specific application or FTL, if the average buffer write ratio is known, we can quickly get a good estimate of the lifetime outcome.

We illustrate the final state of the simulated device configured as 5%/5% after executing the Alix 8d trace in Fig. 3. The surface of the light grey area on the left represents the total SLC-mode lifetime that could be exploited by Phœnix, in this case roughly half of the full potential translating in a 12% improvement. While the current improvement might be humble, we know that more variability in the block endurance amplifies the potential of our scheme, which will inevitably be happening with the future technology nodes.

V. Previous Work

So far, most of the efforts invested to extend the lifetime focus on reducing garbage collection overhead or improving the wear-leveling techniques. Each of those techniques that uses an SLC-mode buffer and MLC data can benefit from our technique.

To the best of our knowledge, the only proposal that extends flash lifetime reusing bad blocks is provided by Wang and Wong [13]. The authors made the observation that when a block is considered as bad, most of its pages are still healthy. They propose to combine the healthy pages of a set of bad blocks together to form a smaller set of virtually healthy blocks. The lifetime of the device is then extended for a reasonable cost, although it not clear by how much.

VI. Conclusion

NAND flash cell storage reliability becomes challenging as cells get smaller and more bits are written to them. We presented Phœnix, a technique that revives bad blocks using the fact that MLC blocks becoming unreliable can still reliably be used to store a single bit per cell. This technique does not cause any direct or indirect extra cost: interestingly, it even reduces slightly the wear-leveling overhead. Phœnix can easily be implemented on top of any existing hybrid FTL and does not require any additional resource—hence, any lifetime benefit comes for free. Using actual flash chips characteristics, we showed up to 17% lifetime extension and we are convinced that future chip technology with higher variance in the block endurance will significantly amplify the advantages of the presented contribution.

References

[1] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *USENIX conf. on File and Storage Technologies*, San Jose, CA, Feb. 2012.

[2] X. Jimenez, D. Novo, and P. Ienne, "Software controlled cell bit-density to improve NAND flash lifetime," in *Design Automation Conf.*, San Francisco, California, USA, Jun. 2012.

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Trans. Consumer Electronics*, vol. 48, no. 2, pp. 366–75, May 2002.

[4] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embedded Computing Systems*, vol. 6, no. 3, Jul. 2007.

[5] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: Locality-aware sector translation for NAND flash memory-based storage systems," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 36–42, Oct. 2008.

[6] H. Cho, D. Shin, and Y. I. Eom, "KAST: K-associative sector translation for NAND flash memory in real-time systems," in *Design Automation and Test in Europe*, Nice, France, Apr. 2009, pp. 507–12.

[7] M.-L. Chiao and D.-W. Chang, "ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation," *IEEE Trans. Computers*, vol. 60, no. 6, pp. 753–66, Jun. 2011.

[8] L.-P. Chang, "A hybrid approach to NAND-flash-based solid-state disks," *IEEE Trans. Computers*, vol. 59, no. 10, pp. 1337–49, Oct. 2010.

[9] S. Im and D. Shin, "ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer," *Journal of Systems Architecture*, vol. 56, no. 12, pp. 641–53, Dec. 2010.

[10] J.-W. Park, S.-H. Park, C. C. Weems, and S.-D. Kim, "A hybrid flash translation layer design for SLC-MLC flash memory based multibank solid state disk," *Microprocessors & Microsystems*, vol. 35, no. 1, pp. 48–59, Feb. 2011.

[11] Micron. (2010, Oct.) Bad block management in NAND flash memory. [Online]. Available: http://www.micron.com/products/support/technical-notes/

[12] K. Bates and B. McNutt. (2007, Jun.) OLTP application I/O. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[13] C. Wang and W.-F. Wong, "Extending the lifetime of NAND flash memory by salvaging bad blocks," in *Design Automation and Test in Europe*, Dresden, Germany, Mar. 2012, pp. 260–63.